AD-A236 723

NRL DOCUMENT: 0926-129

(1)

INTERACTIVE GRO/OSSE REDUCTION ENVIRONMENT (IGORE) SOFTWARE REQUIREMENTS SPECIFICATIONS

GAMMA RAY OBSERVATORY

ORIENTED SCINTILLATION SPECTROMETER EXPERIMENT

DTIC
ELECTE
JUN 1 3 1991
S D
D

Northwestern University
Evanston, IL

N00173-85-C-2501

Release Version 1.0

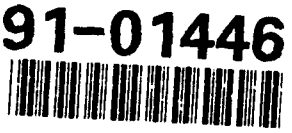
Release Date: August 12, 1988

Author: David Grabelsky

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

91 6 6 061

91-01446



DTIC COPY INSPECTED 5

For
I&I
ed

per AD-A236286

Availability Codes
Dist | Avail and/or Special
A-1

1 INTRODUCTION

1.1 Purpose

This document identifies the software requirements of the interactive data analysis system to be used for the reduction and analysis of OSSE science data. OSSE science data is the output of the OSSE production data analysis system. The interactive analysis system will be known as Interactive GRO/OSSE Reduction Environment, or IGORE for short.

1.2 Scope

The requirement specifications delineate the functional requirements IGORE. IGORE will provide a fully interactive environment within which individual scientists may reduce and analyze data using a variety of scientific programs and interactive manipulations of the data. The main interface between the user and the operations of IGORE will be a "command shell" which will include a programmable, interactive command language. The user will be able to extend IGORE's inventory of data reduction tasks by writing programs either in IGORE's native language or in FORTRAN or other available VAX languages (e.g., C, etc.).

IGORE will consist of five basic components: the High Level Command Language (HLCL)/Command Shell; the Input/Output (I/O) Module; the Data Base Management System (DBMS) Module; the Applications Module; and the IGORE Condition Handler. Each of these will be described in this document. Detailed requirements for the DBMS and Applications Modules are beyond the scope of this document.

Maintenance and updating of this document will be concurrent with the development of the IGORE design document, and will be under configuration control.

1.3 Applicable Documents

The documents listed below will be referred to as needed in this document; they will be referenced according to their number in the list.

1. Data Analysis System Requirements Specification; Gamma Ray Observatory Oriented Scintillation Spectrometer Experiment. Author: Mark S. Strickman.

2. Preliminary Data Analysis Plan; Gamma Ray Observatory Oriented Scintillation Spectrometer Experiment. Author: Mark S. Strickman.

3. Spectral Data Base (SDB) User's Guide; OSSE Software Library; Gamma Ray Observatory Oriented Scintillation Spectrometer Experiment. Authors: Rod S. Hicks, Nina M. Sweeney, and Jack D. Daily.

4. Fit Data Base (FDB) User's Guide; OSSE Software Library; Gamma Ray Observatory Oriented Scintillation Spectrometer Experiment. Author: David Kuo.

1.4 List Of Acronyms

AFE	Applications Frontend
AOE	Abort On Error
COW	Continue On Warning
DBMS	Data Base Management System
EMS	Eat My Shirt
FDB	Fit Data Base
GRO	Gamma Ray Observatory
HLCL	High Level Command Language
ICH	IGORE Condition Handler
IGORE	Interactive GRO/OSSE Reduction Environment
IPOA	Indiscriminate Proliferation Of Acronyms
LOA	List Of Acronyms
NRL	Naval Research Lab
NU	Northwerstern University
OSSE	Oriented Scintillation Spectrometer Experiment
PIF	Program Interface
SDB	Spectral Data Base
UIF	User Interface

2 OVERVIEW OF OSSE SCIENTIFIC DATA ANALYSIS

Documents [1] and [2] give summaries of the scientific goals of OSSE, and of the various, related analyses strategies associated with them. This section first describes the OSSE data types and data bases available, lists rudimentary data analysis tasks, then outlines the basic structure of IGORE by summarizing IGORE's five major components and the interfaces connecting them. Detailed requirements for the interfaces and the components are given in section 4.

2.1 OSSE Data Types And Data Bases

All OSSE data types have the same general structure, consisting of a structured header and an associated data vector. The data vector contains the actual data values associated with the particular data type. The header contains supplementary information associated with the particular data type, including pointers to the data vectors and how to interpret the data vector.

All OSSE data types are stored in associated data bases. There are two basic modes of storeage and management of these data bases. In the first mode, the data vectors, with or without the associated headers, are stored in a data base and managed by a set of dedicated routines which are customized for the particular data type. The main purpose of this mode is to provide access to the actual data involved in the data analysis. In general, only a subset of all archived data of a given type will be resident in this mode of data base at any one

time.

In the second mode, the headers only are stored and managed by a general purpose relational data base management system. The main purposes of this mode are: 1) to provide on line access to sufficient information about all archived data to locate any required data that is not already on line; 2) to provide access to any header quantities required in the data analysis; and 3) to provide pointers, based on header-quantity selection criteria, to data vectors in data bases of the first mode that are already on line.

The following subsections list each OSSE data and its associated data base(s). Each data type listed shall be accessible to the data analysis system. Each can serve as input and/or output in the processing, i.e., access shall not be restricted as exclusively input or output for any of the OSSE data types. IGORE shall accomodate each data type.

2.1.1 SDB Data

2.1.1.1 Data Format

SDB data consist of a structured header and vector of data values. The data and header structures are defined in document [2].

2.1.1.2 Data Bases

Two data bases will be associated with the SDB data. One will contain only headers of all available SDB data. This data base will be managed by a general purpose relational data base management system, and the entire data base will be online. The headers will include pointers to the actual spectral data contained in the other associated data base.

The other SDB data base will contain SDB data in its complete format, i.e., headers and spectral data. This data base will be managed by a dedicated set of routines specific to SDB data. A description of these routines is given in document [3]. Only spectral data under current scientific analysis will be online at a given time; this includes minimally two weeks worth of pointed observations plus SDB data generated by the scientist in the course of the analysis.

Implementation of any auxiliary data base containing a subset of spectral header quantities and managed by the general purpose relational data base management system shall be possible. This data base would be associated with the same spectral data as full-header data base from which it is derived.

2.1.2 Time Series Data

2.1.2.1 Data Format

Time series data consist of a structured header and a vector of data values. The data and header structures and are given in document [xx.xx].

2.1.2.2 Data Bases

Two data bases will be associated with the time series data. The management of the two data bases is the same as that for the SDB data bases described above (section 2.1.1.2). The header structure of time series data will be different from the SDB header structure (see document [xx.xx]), but will still contain pointers to the actual data vectors contained in the actual data files.

2.1.3 FDB Data

2.1.3.1 Data Format

FDB data consist of a structured header and structured data. The structure and contents of each is given in document [4].

2.1.3.2 Data Bases

The FDB data will be managed by a set of dedicated routines specific to FDB data. A description of these routines and the structure of FDB data are given in document [4].

2.1.4 Pulsar Data

2.1.4.1 Data Format

Pulsar data consist of structured pointers and data stored in telemetry archive media or in dedicated files. Details are given in document [ww.xx].

2.1.4.2 Data Bases

Two data bases will be associated with pulsar data. One, consisting only of headers, will be the similar to the header data base associated with SDB data described above (section 2.1.1.2). This data base will be managed by the general purpose relational data base system. The headers contain pointers to the actual data files.

The other associated data base will consist of telemetry archive or dedicated pulsar data media.

2.1.5 Instrument Model Data

2.1.5.1 Data Format

Instrument model data will be stored using the FDB and SDB described above. It does not comprise a separate data file.

2.1.5.2 Data Bases

The instrument model data will be a combination of SDB and FDB data, and be managed both by the general purpose relational data base system (headers only), and by dedicated routines specific for SDB and FDB data, respectively. The specific breakdown of the instrument model data base is as follows:

1. Calibration data - managed same way as SDB data.

2. Instrument response data - managed similarly to FDB data.

3. Instrument state data - in headers managed by general purpose DBMS.

2.2 Rudimentary Data Analysis Tasks

A variety of data analysis strategies will be used by each scientist, but certain rudimentary data analysis tasks will be used on a fairly routine basis. The following list identifies and briefly describes these tasks.

1. Detector efficiency determination -

   This will involve as input detector temperature at time of observations, the DTWs used, and the instrument calibration results. The temperature information will be in the SDB header, the DTW information and the calibration results in the Instrument Model Data Base. The inputs will be passed to an application program that determines the efficiency.

2. Spectral gain correction -

   This will involve as input fit results from Co60 memory of the relevant spectra and a standard channel configuration from the Instrument Model Data Base or, optionally, from the scientist. The fit results will be retreived from the headers of the relevant spectra. An application program will produce resampled spectra. Gain correction will be performed either before or after summation or subtraction of spectra, depending on the time scale over which spectra are to be summed or subtracted.

3. Spectral background subtraction -

   This will involve the SDB spectral data and headers as input to a specific applications program. The gain of the operand spectra must be the same; this will be assured by using spectra taken over short time scales or by first applying gain correction as described in item (2).

4. Spectral summation -

   This will involve the SDB spectral data and headers as input to a specific applications program. The gain of the operand spectra must be the same; this will be assured by using spectra taken over short time scales or by first applying gain correction as described in item (2).

5. Instrument response determination -

   This involves as input pointing information from the spectral headers and instrument angular response from the Instrument Model Data Base. The inputs are used by an applications program to determine the instrument response for the viewing angle with respect to the astrophysical target source.

6. Spectral line and continuum fitting -

   This involves as input SDB spectral data; either unprocessed science data or background-subtracted, gain-corrected, summed spectra. The fit is performed by an applications program on count-space spectra.

7. Spectral deconvolution -

   This involves as input SDB spectral data, processed as in the preceding list item, the instrument response as determined in item (5), and source spectrum model parameters from a file or from the interactive user. The incident photon spectrum is calculated by an applications program.

8. Diffuse source modeling -

   This is similar to the preceding item with the addition of source spatial distribution model parameters with the input model spectrum.

9. Epoch folding -

   This will involve as input pulsar data and an estimate of pulse period. An applications program will resample the data in time bins relative to the input pulse phase. Barycenter arrival times of time-tagged signals will be computed from the barycenter vector and source direction, both contained in the spectral header.

2.3 The Role Of IGORE In OSSE Data Analysis

IGORE will provide the main user interface, in the form of an interactive environment, to the package of analysis programs and tools required to analyze OSSE data. The scientific user must have a large library of routine analysis tools available as well as the means to execute these programs, examine intermediate results, add new programs as needed, and execute sequences of programs. Passing data and parameters between different programs must be simple, efficient, and fast. Because of the large number parameters often associated with various analyses, e.g., SDB header quantities, source models, results of previous fits, etc., IGORE must also provide a convenient method for managing collections of parameters. The following list is a generalized overview of the facilities that IGORE will provide.

1. Provide a high level command language (HLCL) for running individual programs, executing strings of programs, examining results, interactively manipulating data and parameters, and writing programs, procedures, and functions.

2. Provide as native capabilities the specific tasks and applications described in the preceding section (2.2).

3. Provide and interface to all the data and data bases via dedicated input and output routines and the data base management system.

4. Provide a simple and robust FORTRAN or FORTRAN-compatible interface to allow users to incorporate new FORTRAN routines into the IGORE environment; this includes the capability of passing data and control parameters in and out of the user routines.

2.3.1 Summary Of The Major Components Of IGORE

IGORE will consist of five components referred to here as modules. Each will group together routines which perform logically related tasks and functions. A brief description of each is given in the list below. The next section (2.4.2) identifies the required interfaces.

1. High Level Command Language (HLCL)/Command Shell. The functions of the HLCL/Command Shell are:

   - o Main user interface to all other components of IGORE.
   - o Interactive command/data manipulation language.
   - o Provide native variables for all OSSE data types.
   - o Executive for applications.
   - o Programming language.
   - o Logging and journaling.
   - o Access to VMS and DCL.

2. Input/Output (I/O) Module. The functions of the I/O module are:

   - o Access to the science data bases (except those managed directly by the DBMS).
   - o Contains specialized routines for dedicated read/write access to specific data types: SDB files and FDB files.
   - o Contains SDB editors.
   - o General purpose disk and tape I/O routines.

3. Data Base Management System (DBMS) Module. The functions of the DBMS Module are:

   - o Contains the general purpose reltional DBMS program.
   - o Provides a frontend to DBMS for interfacing DBMS interactive commands with IGORE HLCL.
   - o Manages headers of all SDB, time series, pulsar and instrument state data.
   - o Provides pointers to the SDB, time series, and pulsar data vectors.
   - o Can run DBMS in standalone mode.

4. Applications Module. The functions of the Applications Module are:

   - o Contains all scientific analysis programs for performing task listed above in section 2.2.
   - o Interfaces applications programs to interactive level via frontend programs.

5. IGORE Condition Handler. The function of the IGORE Conditon Handler (ICH) is to provide a central error reporting and recovery system for all actual or potential errors that may occur during execution of any program that is part of IGORE. The IGORE Condition Handler will also provide a central message reporting facility for warning and status messages.

2.3.2 Summary Of Required Interfaces

IGORE interfaces shall be divided into two categories: User Interfaces (UIFs) and Program Interfaces (PIFs). PIFs shall connect IGORE system modules for transfer of data and activation of program execution. UIFs shall function on a DEC VT100 terminal with retrographics (or compatible type) and provide the user with the means for interactive data and command input. Data refers to specified data types.

2.3.2.1 IGORE Program Interfaces

The following list identifies which modules shall be connected via a standard PIF. Each item in the list further specifies which OSSE data types shall be passed across the interface. Detailed requirements for each interface are given in section 4.1.

1. HLCL-FORTRAN Applications Interface. This interface shall connect the HLCL/Command Shell with the Applications Module. All OSSE data types shall be passed across this interface, in both directions.

2. HLCL-I/O Interface. This interface shall connect the HLCL/Command Shell with the I/O Module. All OSSE data types except pulsar data shall be passed across this interface, in both directions. This interface shall also have the capability of passing all VAX FORTRAN or FORTRAN-compatible data types and structures, in both directions.

3. HLCL-DBMS Interface. This interface shall connect the HLCL/Command Shell with the DBMS Module. All data types managed by the DBMS shall be passed across this interface; passing shall be in both directions. This interface shall also have the capability of passing all VAX FORTRAN or FORTRAN-compatible data types and structures, in both directions.

4. FORTRAN Applications-I/O Interface. This interface shall connect the I/O Module with the Applications Module. All OSSE data type shall be passed across this interface, in both directions. This interface shall also have the capability of passing all VAX FORTRAN or FORTRAN-compatible data types and structures, in both directions.

5. FORTRAN Applications-DBMS Interface. This interface shall connect the Applications Module with the DBMS Module. All data types managed by the DBMS shall be passed across this interface; passing shall be in both directions. This interface shall also have the capability of passing all VAX FORTRAN or FORTRAN-compatible data types and structures, in both directions.

6. IGORE Condition Handler Interface. The ICH interface shall consist of a VMS System call to establish the ICH as the condition handler. This call shall be made automatically as part of the HLCL-Modules interface support (i.e., any of items 1-3 above).

2.3.2.2 IGORE User Interfaces

All IGORE UIFs shall function on a DEC VT100 terminal with retrographics or compatible type. The following list identifies the UIFs which shall be available in IGORE. Each item in the list also specifies what data types shall be passed across the interface. Detailed requirements for UIFs are given in section 4.2.

1. User-HLCL Interface. This interface shall be the main UIF to the interactive environment provided by IGORE. The user shall invoke all interactive capabilites provided by the HLCL/Command Shell by entering commands and data at the terminal keyboard. All VAX FORTRAN or FORTRAN-compatible data types and structures shall be passed across this interface, in both directions.

2. User-FORTRAN Applications Interface. This interface shall consist of any interactive portion of any applications program. Detailed requirements for interactive I/O of a given applications program shall fall within the scope of the requirements for that program. IGORE will provide support for jounaling user I/O on this interface.

3. User-DBMS Interface. This interface shall consist of the interactive query language of the DBMS. Detailed requirements for this interface fall within the scope of the DBMS requirements. General requirements for the UIF of the DBMS are given in section 4.2.

3 DESIGN CONSTRAINTS

3.1 VAX/VMS Based

IGORE must run under VAX/VMS. IGORE system design should make full use of the VMS system and environment whenever it is advantageous to do so.

3.2 Default User Terminal

The default user terminal for IGORE shall be a DEC VT100 with retrographics or compatible type. IGORE's full range of operations shall be available to any user interfaced to IGORE via the default terminal.

3.3 Computer Resources

The IGORE executable image will be run by several (6-10) users at any one time. The baseline computer systems are: a VAX 11/785 with 12 megabytes of memory and 4 gigabytes of disk space at the Naval Research Lab (NRL), and a VAX 11/750 with 15 megabytes of memory and 1.4 gigabytes of disk space at Northwestern University (NU). IGORE will be sufficiently economical as not to overwhelm either of these systems.

The IGORE executable image shall be a VMS installed shareable image (shared shareable).

4 INTERFACE REQUIREMENTS

IGORE shall provide support for interfaces in two catagories: user interfaces (UIFs), and program interfaces (PIFs). PIF requirements are detailed in the next subsection (4.1) and UIF requirements are detailed in the following subsection (4.2)

4.1 IGORE Program Interfaces

IGORE program interfaces shall connect individual program units which require interchange of data and/or activation of execution of one program unit by the other. Requirements for the interfaces between the HLCL and any of the other modules of IGORE are given in sections 4.1.1-4.1.4. Requirements for module-module interfaces that bypass the HLCL are described in sections 4.1.5-4.1.6.

4.1.1 Applications Frontend For General HLCL-Modules Interface

The interface between the HLCL and any program unit in any other module of IGORE shall consist of a frontend routine, referred to as the applications frontend or AFE. The AFE shall contain two sub-interfaces: one to the HLCL and the other to the specific program unit being interfaced. All program units to be interfaced to the HLCL shall be referred to as "applications," regardless of their function and the module in which they are contained.

All applications shall require an AFE to interface to the HLCL. The function of the AFE shall be to activate execution of the application and to tranfer data, if any, between the AFE's two interfaces. Each AFE shall have the same general structure, consisting of calls to AFE support routines and a call to the application being interfaced. The only distinctions between different AFE's shall be the specific parameters passed, the specific application called, and custom settings of AFE control parameters. The general structure of the AFE shall be described by a template; a specific AFE shall be built by placing the names of the parameters to be passed and the name of the application to be activated in the template, and optionally modifying default AFE control parameters.

The list below specifies the requirements for support of the general AFE; these requirements are general and apply to the interfaces between the HLCL and each of the other modules of IGORE. Specific requirements for the other modules of IGORE specify the actual OSSE data types that must be accomodated across each interface; these requirements are given in the subsections 4.1.2-4.1.4.

1. The AFE shall consist minimally of code that establishes the IGORE condition handler (section 5.5) and then activates the application according to the specifications of the application. Upon completing execution, the application shall return to the AFE. The AFE shall then de-establish the ICH.

2. The AFE shall be activated only by a native command issued from the HLCL. The activating HLCL command shall be issued either by an interactive user or from within an HLCL program than includes the command. Upon completing execution, the AFE shall return to the HLCL.

3. All input and output to be transferred between the HLCL and the application shall be passed as parameters in the call from the HLCL to the AFE. The list of these parameters shall comprise the data interface between the HLCL and the AFE. Handshaking shall be used on this interface; see item 5. The parameter list shall be divided into "required" parameters and "optional" parameters. Required parameters are those which must be supplied explicitly in the call to the AFE; optional parameters are those which, if not supplied explicitly, shall be set to previously assigned default values. Classification of parameters as either required or optional shall be determined statically within the AFE when the AFE is coded. The specific classification for a given application shall be a programming option.

4. The data interface between the AFE and the application shall consist of a set of FORTRAN or FORTRAN-compatible data structures, each of which corresponds to one of the parameters passed by the HLCL. No handshaking shall be used on this interface. The application shall expect to receive as input and/or transmit as output the data types corresponding to the FORTRAN or FORTRAN-compatible structures defined in the AFE. Accordingly, no handshaking shall be established directly between the HLCL and the application.

5. HLCL-AFE handshaking. Utility routines shall be provided which transfer HLCL data structures to corresponding FORTRAN or FORTRAN-compatible data structures and vice versa. The specific functions of the transfer routines shall be:

a) To check that all parameters have been supplied. If any required parameters have not been supplied, the AFE shall prompt for them; if any optional parameters have not been supplied, the AFE shall assign default values to them. This function shall apply only to HLCL-to-FORTRAN structure transfers.

b) To check that the types and sizes of all pairs of HLCL-FORTRAN variables match. Any mismatch, with the exceptions noted below in item 6, shall constitute an error. In the case of such an error, an exception shall be signaled causing the IGORE condition handler to be invoked with an abort on error condition (section 5.5.1.3); i.e., the AFE shall abort without activating the application and an appropriate error message shall be reported. This function shall apply to transfers in both directions, but the check shall be made prior to activating the application.

c) To make the actual transfer of data between the HLCL parameters and the FORTRAN of FORTRAN-compatible structures. This function shall apply to transfers in both directions.

6. The utility routines shall provide a type conversion mode for the following mismatched types (scalars only):

REAL*8 - INTEGER*4
REAL*8 - INTEGER*2
REAL*8 - REAL*4
REAL*4 - INTEGER*4
REAL*4 - INTEGER*2
INTEGER*4 - INTEGER*2

Implementation of selected modes shall be made on a per AFE-basis as a programming option. Each type conversion mode shall be enabled or disabled according to a bit in a type conversion flag. Type conversion mode enabling shall be possible only for those modes whose associated bits can be manipulated dynamically. A mask shall determine which bits may be set dynamically. The default flag shall disable all type conversion modes, and the default mask shall prohibit dynamic manipulation of any bits in the flag. Replacement of the default mask with one that permits dynamic manipulation of selected bits in the flag shall be possible only by explicitly hardwiring the new mask into the AFE code. The HLCL shall provide an interactive command for setting the type conversion mode flag. Attempting to manipulate a disabled bit with this command shall result in a warning message (continue on warning condition; see section 5.5.1.3).

7. Input to the application shall be transferred from the input HLCL parameters to the corresponding FORTRAN or FORTRAN-compatible data structures prior to activating execution of the application.

8. Output from the application shall be transferred from the FORTRAN or FORTRAN-compatible data structures to the corresponding output HLCL parameters upon return from the application to the AFE and prior to return from the AFE to the HLCL.

4.1.2 HLCL-FORTRAN Applications Interface

The HLCL-FORTRAN Applications Interface shall be able to pass all OSSE data types referred to in section 2.1. In addition, this interface shall be able to pass all VAX FORTRAN or FORTRAN-compatible data types, including structures. Detailed requirements for each data type are as follows.

4.1.2.1 SDB Data

1. Each SDB-format spectrum shall consist of a structured header and a vector containing the spectral data.

2. The HLCL shall provide the following native variables for SDB-format spectra:

   - o header plus spectrum vector
   - o spectrum vector only
   - o header only

3. FORTRAN applications shall access SDB-format spectra as any one of the FORTRAN or FORTRAN-compatible data types:

   - o structure plus vector for header plus spectrum
   - o vector for spectral data only
   - o structure for header only

4. Both the HLCL and FORTRAN applications shall have read and write access to individual header variables and data words and to spectral data vectors.

5. Special purpose header structures and/or data vector structures shall be accomodated by declaring new FORTRAN structures in the applications and by modification of HLCL tables.

6. No specific handshake shall be required between the HLCL and the application to determine the variable type being passed. The application definition shall specify each variable type

it its passed parameter list, and the HLCL will expect to send and/or receive the appropriate type in each case.

4.1.2.2 Time Series Data

Time series data are similar to SDB data but with a different header structure. The interface requirements are identical to those for SDB data.

4.1.2.3 FDB Data

1. Model fit results shall be stored in the FDB format.

2. These shall be represented in the HLCL as a structured variable containing the FDB header information from the spectrum layer downward and an array of structured variables, each element of which contains fit parameter header information and the fit data.

3. These shall be represented in the application as FORTRAN structured variables arranged in a similar fashion to the HLCL variables described above.

4. The HLCL and application shall have access to individual header and data quantities.

5. Header and data quantities shall be kept together in both the HLCL and the application.

6. No specific handshake shall be required between the HLCL and the appliction to determine the variable type being passed. The application definition shall specify each variable type it its passed parameter list, and the HLCL will expect to send and/or receive the appropriate type in each case.

4.1.2.4 Pulsar Data

1. The HLCL shall not be required to handle pulsar data directly.

2. Pulsar data shall be read only by FORTRAN or FORTRAN-compatible application programs that shall run under IGORE or independently.

3. The HLCL shall handle, via the DBMS interface, pointers to pulsar data extracted from the collection data base.

4. Pulsar data pointers shall consist of structured data resembling SDB headers and containing the same information relevant to the interval of pulsar data.

5. Pulsar data pointers shall be handled by the HLCL and applications in a manner identical to SDB headers.

6. In addition, pulsar data pointers shall contain pointers to records, files, and archive media containing the desired pulsar data. The pointed archive media shall be either the telemetry archive as produces by the production analysis process, or specially produced pulsar media. Applications shall read one or both of these formats.

4.1.2.5 Instrument Model Data

1. Instrument model data shall be stored in SDB and FDB formats.

2. Requirements for instrument model data stored in SDB format shall be identical to those for SDB data given above (section 4.1.2.1).

3. Requirements for instrument model data stored in FDB format shall be identical to those for FDB data given above (section 4.1.2.3).

4.1.2.6 Auxiliary Data

1. Auxiliary data types include all VAX FORTRAN or FORTRAN-compatible types and structures.

2. The HLCL shall be able to access all VAX FORTRAN or FORTRAN-compatible data types as native variables.

3. The HLCL shall maintain a table of allowed structures. These shall be referred to as "known structures." The HLCL shall be able to access all known structures and their respective constituent fields as native variables.

4. FORTRAN applications shall access all auxiliary data as compiled FORTRAN data types or structures.

5. Extension of the inventory of known structures shall be possible by compiling new definitions in a FORTRAN application and its associated AFE, and by updating the appropriate HLCL table.

4.1.3 HLCL-I/O Interface

The HLCL-I/O Interface shall be able to pass all OSSE data types referred to in section 2.1 except pulsar data. In addition, this interface shall be able to pass all VAX FORTRAN or FORTRAN-compatible data types, including structures. Detailed requirements for each data

type are as follows.

4.1.3.1 SDB Data

Each dedicated routine for access of SDB files (SDB Library) shall be interfaced to the HLCL via an AFE.

4.1.3.2 Time Series Data

Time series data files (headers plus data vectors) shall be managed by the same dedicated routines that access other SDB data. The requirements for HLCL-I/O Module interface for time series data are identical to those for SDB data specified above (section 4.1.3.1).

4.1.3.3 FDB Data

Each dedicated routines for access of FDB files (FDB Library) shall be interfaced to the HLCL via an AFE.

4.1.3.4 Pulsar Data

Pulsar data shall not be passed across this interface.

4.1.3.5 Instrument Model Data

1. Requirements for instrument model data stored in SDB format shall be identical to those for SDB data given above (section 4.1.3.1).

2. Requirements for instrument model data stored in FDB format shall be identical to those for FDB data given above (section 4.1.3.3).

4.1.3.6 Auxiliary Data Types

1. The AFE for any I/O routines shall be able to pass auxiliary data types across this interface.

2. The requirements for passing auxiliary data across this interface are identical to those specified in section 4.1.2.6.

4.1.4 HLCL-DBMS Interface

The HLCL-DBMS Interface shall be able to pass all OSSE data types managed by the general purpose relational DBMS. These data consist of all OSSE spectral headers for spectral, time series, and pulsar data, and instrument state data. Any future data bases managed by the DBMS shall be accessible across this interface. In addition, this interface shall be able to pass all VAX FORTRAN or FORTRAN-compatible

data types, including structures.

This interface shall be used access the DBMS via HLCL commands. Three modes of access to the DBMS shall possible: 1) invoking specific DBMS functions, 2) invoking the DBMS native interactive environment (query language and/or menu facility), and 3) invoking the DBMS in standalone mode. In modes (1) and (2), it shall be possible to pass output from the DBMS directly back to the calling program across the interface. Status codes and error conditions detected in DBMS routines shall be returned across this interface to the calling program. These shall be examined and, if necessary, an IGORE condition shall be signaled (see section 5).

In mode (3), the output from the DBMS shall be accessible outside of the DBMS only as disk file; i.e., it shall not be possible to pass output generated by the DBMS in standalone mode directly back to the program that invoked this mode. Modes (2) and (3) are actually part of the DBMS user interface; mode (1) shall be considered part of the DBMS interface if the calling program includes user I/O relevant to the DBMS call (see section 4.2.3).

Detailed requirements for this interface for each OSSE data type are as follows.

4.1.4.1 SDB Data

1. Only SDB headers shall be accomodated.

2. All DBMS routines for access of SDB headers shall be interfaced to the HLCL via an AFE.

4.1.4.2 Time Series Data

1. Only time series data headers shall be accomodated.

2. Time series data headers are similar to SDB headers and a data base consisting of time series headers only shall be managed by the DBMS. The requirements for HLCL-DBMS Module interface for time series data headers are identical to those for SDB data specified above (section 4.1.4.1).

4.1.4.3 FDB Data

FDB data shall not be passed across this interface.

4.1.4.4 Pulsar Data

1. Only pulsar data headers shall be accomodated.

2. All DBMS routines for access of pulsar headers shall be interfaced to the HLCL via an AFE.

4.1.4.5 Instrument Model Data

1. Only instrument state data contained in SDB header format and managed by the DBMS shall be accomodated.

2. Requirements for these data shall be identical to those for SDB data given above (section 4.1.3.1).

4.1.4.6 Auxiliary Data Types

1. All AFEs to any DBMS functions shall be able to pass auxiliary data types across this interface.

2. The requirements for passing auxiliary data across this interface are identical to those specified in section 4.1.2.6.

4.1.5 FORTRAN Applications-I/O Interface

The FORTRAN Applications-I/O Interface shall follow the standard VAX FORTRAN calling conventions for functions and subroutines. The specific format of the calls and the required parameter lists shall follow the documented guidelines for the FORTRAN interface of the particular I/O routines being called. The guidelines for SDB I/O are given in documents [3]; the guidelines for FDB I/O are given in document [4]. This interface shall be able to pass all OSSE data types referred to in section 2.1. In addition, this interface shall be able to pass all VAX FORTRAN or FORTRAN-compatible data types, including structures, and any other data types allowed by the FORTRAN interface of the particular I/O routines being called.

4.1.6 FORTRAN Applications-DBMS Interface

The FORTRAN Applications-DBMS Interface shall follow the standard VAX FORTRAN calling conventions for functions and subroutines. The specific format of the calls and the required parameter lists shall follow the documented guidelines for the FORTRAN interface of the DBMS program. This interface shall be able to pass all OSSE data types managed by the DBMS referred to in section 2.1. In addition, this interface shall be able to pass all VAX FORTRAN or FORTRAN-compatible data types, including structures, and any other data types allowed by

the FORTRAN interface of the DBMS.

This interface shall be used for calls from FORTRAN applications to the DBMS. Three modes of access to the DBMS shall possible: 1) invoking specific DBMS functions, 2) invoking the DBMS native interactive environment (query language and/or menu facility), and 3) invoking the DBMS in standalone mode. In modes (1) and (2), it shall be possible to pass output from the DBMS directly back to the calling program across the interface. Status codes and error conditions detected in DBMS routines shall be returned across this interface to the calling program. These shall be examined and, if necessary, an IGORE condition shall be signaled (see section 5).

In mode (3), the output from the DBMS shall be accessible outside of the DBMS only as disk file; i.e., it shall not be possible to pass output generated by the DBMS in standalone mode directly back to the program that invoked this mode. Modes (2) and (3) are actually part of the DBMS user interface; mode (1) shall be considered part of the DBMS interface if the calling program includes user I/O relevant to the DBMS call (see section 4.2.3).

4.2 IGORE User Interfaces

User interfaces shall function on a VT100 terminal with Retrographics (or compatible type) and provide the user with the means to input commands and data to IGORE and receive messages, graphics, and other printable output from IGORE. IGORE shall provide three basic UIFs: 1) the HLCL/Command Shell; 2) any interactive portions of any application program that runs under IGORE; and 3) the interactive frontend to the DBMS. These are described below in sections 4.2.1-4.2.3.

4.2.1 HLCL/Command Shell User Interface

The primary interactive language of IGORE shall be referred to as the High Level Command Language (HLCL). The Command Shell shall be the interactive environment within which HLCL commands are issued. The HLCL shall be primarily a command driven language with any menu capabilities layered on top of the command mode of the HLCL. The requirements for the UIF of the HLCL are given in the following list. Functional requirements for the HLCL/Command Shell are given in section 5.1.3.

1. Login. The user shall log in to IGORE by issuing a VMS DCL command to run the IGORE executable file.

2. Logout. The user shall log out of IGORE by issuing an HLCL command that terminates execution of the IGORE executable file.

3. Prompt. After automatic intialization of the environment, the HLCL shall present the user with a command line prompt. It shall then be possible for the user to issue any command at the keyboard.

4. Issuing commands. The user shall issue commands to IGORE by typing the command at the keyboard, terminated by a carriage return. It shall be possible for commands to extend over more than one line by typing a special character before the carriage return that indicates to the HLCL that the command continues on the next line.

5. Parameters. The HLCL shall recognize two types of parameters of programs invoked by HLCL commands: required parameters and optional parameters. Required parameters must be supplied explicitly before IGORE activates the actual program. Optional parameters do not need to be supplied explicitly before IGORE activates the program.

6. HLCL reponse. The HLCL response to a given command shall vary depending on the nature of the command. The HLCL responses shall be as follows:

   - o Valid command, parameterless program. The program invoked by the command shall execute. When execution completes, control shall be returned to the HLCL prompt.
   - o Valid command, required and/or optional parameters. Parameters to programs shall be entered on the command line with the HLCL command that invokes the program. If required parameters are not explicitly supplied with the command, the user shall be prompted to enter them. If optional parameters are explicitly supplied with the command, they shall be passed to the program; if optional parameters are not supplied, default values shall be assigned and passed to the program.
   - o Invalid command. Invalid commands shall be those which violate in some way the syntax of the HLCL. An appropriate message shall be issued to the user in response to any invalid commands, and control shall continue at the HLCL prompt.

7. Abnormal program function. Abnormal execution of any program invoked by an HLCL command shall result in one of two conditions: continue on warning (COW) or abort on error (AOE). See section 5.5 for details on the IGORE condition handler. The result of a COW shall be an appropriate warning message to the user, and continued execution of the program. The result of an AOE shall be an appropriate error message, and aborted execution with return of control to the HLCL prompt. NB: no recovery is possible for VMS access violations; i.e., these will cause IGORE to crash.

8. Status messages. Any program invoked by an HLCL command shall have the capability of issuing informational and/or status messages to the user.

9. Menu mode. Menues shall be available for: 1) driving individual programs through parameters entered into the menus associated with the programs, and 2) for organizing groups of programs related to specific analysis strategies. In the first use, the menu items shall correspond to all parameters of a program that is invoked by an HLCL command. In the second use, the menu items shall correspond to the names of programs that are invoked by HLCL commands; chosing one of them from the menu shall have the same effect as typing the HLCL command. In either use, the menu facility shall be built upon the command mode of IGORE.

10. Journaling. The HLCL shall optionally journal to a journal file all alphanumeric I/O across this UIF. Journaling shall be enabled or disabled according to a switch set with an HLCL command. User input shall be written verbatim to the journal file. Program output shall have a leading comment character inserted before being written to the file. Command entered in menu mode shall be tranlated to their equivalent command mode form before being written to the file.

11. Documentation. Documentation shall consist of an IGORE User's Guide and online help. Online help shall be limited to a brief description of all valid commands and associated parameters. The IGORE User's Guide shall provide a complete description of how to use IGORE, including programming and syntax rules.

4.2.2 Interactive Applications User Interface

The interactive applications UIF shall be defined as any portion of a program running under IGORE that issues messages and/or prompts to the user and accepts data input from the user and that bypasses the HLCL in doing so.

All alphanumeric I/O across this interface shall be optionally journaled to the same journal file used for HLCL journaling (if enabled). Graphics I/O across this interface shall not be journaled. Journaling shall be enabled or disabled according to a switch set at the HLCL level. All user input shall be written to the journal file verbatim. All message output from the program shall have a leading comment character inserted before being written to the journal file.

Journaling on this interface shall be available only for programs which make use of the standard VAX terminal I/O instructions of the language in which the program is written (i.e., in FORTAN: READ(5,...), WRITE(6,...)).

4.2.3 DBMS User Interface

Two modes of DBMS UIF shall be available. These are described in the following subsections. Specific functions of the DBMS available in IGORE are described in sections 5.1.3.4 and 5.3.

4.2.3.1 External Program Access

This mode of access to the DBMS shall be made through the DBMS-supplied interface for calls by external programs and shall be considered an interactive UIF only if: a) the specific interactive I/O of the external calling program is relevant to DBMS functions (access types (1) and (2) below), or b) the external program invokes interactive capabilities native to the DBMS (access type (3) below). The external calling program shall either be an applications program or an HLCL command. The UIF requirements in each case are given in sections 4.2.1 and 4.2.2, respectively.

Three types of user access to the DBMS across its external program interface shall be possible:

1. Hardwired commands. These shall be commands in the HLCL, menu choices from HLCL menus, or menu-type choices from an interactive program. Hardwired commands shall not be dynamic; i.e., they shall invoke specific DBMS functions. The output from the DBMS for such commands shall not be dynamic; i.e., the type of output expected shall also be hardwired into the command.

2. Interactive DBMS commands entered from the calling program. These shall be commands in the native query language of the DBMS that are entered by the user in the calling program and then passed to the DBMS. These commmands shall be dynamic; i.e., the user shall be able to enter any valid DBMS queries from within the HLCL or any interactive program capable of accepting appropriate input strings. The output from the DBMS for such commands shall be dynamic; i.e., the type of output shall depend on the command but shall, in any case, be one of the data types supported on the DBMS-external program interface.

3. Interactive DBMS environment access. This shall be DBMS access through any native interactive DBMS capabilities that can be invoked by calls across the DBMS-supplied external program interface. These capabilities include the DBMS query language and menu facilities. This mode is dynamic. Output from the DBMS shall depend on the commands but shall, in any case, be one of the data types supported on the DBMS-external program interface.

User input to interactive programs shall be journaled (if journaling is enabled) prior to being passed to the DBMS for access types (1) and (2) above. For type (3), only the interactive request to invoke the

DBMS environment shall be journaled; standard IGORE journaling shall not be possible for commands entered in the native environment of the DBMS.

Output from the DBMS shall be any of the data types described for the HLCL-DBMS interface (section 4.1.4) or the Applications-DBMS interface (section 4.1.6). In addition, status and error output from the DBMS shall be checked by the calling program, and if necessary, a condition shall be signaled. Output from the DBMS that is directed to the terminal shall be journaled (if journaling is enabled) after it is received by the interactive program.

4.2.3.2 DBMS Native Interactive Environment

It shall be possible to enter the native interactive environment of the DBMS from the HLCL or any interactive application running under IGORE. The requirements of this interface are within the scope of the DBMS requirements specifications. The data interface between the DBMS and any other modules of IGORE when the DBMS is accessed in this mode shall be disk files. It shall not be possible to pass data directly in either direction between the DBMS and any other module of IGORE when the DBMS is accessed in this mode. Standard IGORE journaling shall not be possible in this mode. Note, however, that if this mode is entered by way of an HLCL command, the command will be journaled.

5 FUNCTIONAL REQUIREMENTS OF IGORE

5.1 High Level Command Language/Command Shell

5.1.1 Definitions And General Considerations

The Command Shell shall be the interface between the main interactive user and IGORE. The Command Shell shall provide the interactive environment within which the user executes programs and commands. The High Level Command Language (HLCL) shall be the interactive language for issuing commands. The requirements for the HLCL user interface are given in section 4.2.1. The HLCL shall also a programming language for creation of new commands, programs, procedures and functions, as well as a data manipultion language for examination and modification of data processing results.

5.1.2 Data Types Allowed

The HLCL/Command Shell shall provide native variables for auxiliary data types (section 4.1.2.6) and all OSSE data types except pulsar data. The following list summarizes how these data types shall be represented in the HLCL

1. SDB data:

    - o structured headers plus data vectors
    - o structure headers only

- o data vectors only

2. Time series data:

   - o same as SDB data but different header structure

3. FDB data:

   - o structured headers containing arrays of structured variables

4. Pulsar data:

   - o structured header similar to SDB headers
   - o variables for actual data not required in HLCL

5. Instrument model data:

   - o combination of SDB and FDB data formats

6. Auxiliary data:

   - o all VAX FORTRAN data types
   - o compiled and linked FORTRAN or FORTRAN-compatible structures

5.1.3 HLCL/Command Shell Functional Requirements

The functional requirements of the HLCL/Command Shell specify both the interactive and system-level functions of the HLCL/Command Shell.

5.1.3.1 General Interactive Functions

The following list specifies the general interactive functions that shall be native to the HLCL/Command Shell.

1. Interactive declaration of native variables. Any number of variables of any type which is native to the HCLC/Command Shell shall be declarable interactively. Each new variable shall have a distinct, user-specified name. The native data types are specified in section 5.1.2.

2. The HLCL shall have read/write access to all its declared variables.

3. Math, logic, string, and character procedures and functions. These shall include all arithmetic and transcendental functions, relational logic, boolean logic, and string and character manipulation available in FORTRAN.

4. Vector and array manipulation. Vectors and arrays of any variable type shall be interactively declarable. Ranges and single elements of vectors and arrays shall be addressible by their indeces. Additional procedures and functions shall provide vector and matrix multiplication and matrix inversion.

5. Programmability. The HLCL shall provide a programming language that allows native capabilities and commands to be collected into program units. Program creation shall either be done in an editor or via an immediate command mode by entering the program directly at the interactive command level. The user shall be able to save programs created in the editor in disk files. The programming language shall have capabilities comparable to FORTRAN.

6. Graphics. The HLCL shall provide rudimentary interactive graphics routines that operate on native HLCL variables. These routines shall include x-y plots of two HLCL 1-dim vectors, contour plots of an HLCL 2-dim array, cursor positioning and readback, and plot labeling. These graphics routines shall be distinct from more elaborate graphics applications that operate FORTRAN variables internal to the applications.

7. Editor. IGORE shall provide access the VMS text editors (EDT or TPU).

8. Journaling. The HLCL/Command Shell shall provide a utility for automatic journaling of all user I/O across the UIF of the HLCL and any interactive program; user I/O in the native interactive environment of the DBMS shall not be journaled by this procedure. See section 4.2 for restrictions on journaling of user I/O on the UIFs. The journal shall be saved in a text file which may be later read and modified in the editor. It shall also be possible to submit the journal file in a batch mode and have the journaled interactive session replayed in batch mode with restrictions noted section 5.1.3.5 (batch mode processing).

9. Saving and Restoring. The HLCL shall provide a utility for limited saving the current IGORE environment. The saved environment shall include:

   - o all declared HLCL variables
   - o all compiled programs written in the HLCL language
   - o all links to external programs created during the interactive session

The HLCL shall provide a utility for restoring the IGORE environment from the information saved by the save utility.

10. Access to DCL and VMS. The HLCL shall be able to parse and pass DCL commands back to the parent VMS process. The HLCL shall also include an inventory of commands that invoke useful VMS utilities contained in the Runtime Library and the System Utilities Library.

5.1.3.2 Data Analysis Tasks

Routine data analysis tasks shall be included as native functions in the HLCL. These listed and briefly described in section 2.3. They are listed again below with their respective input and output data types and the the interfaces accessed.

1. Detector efficiency determination -

   - Inputs: SDB headers, Instrument state data
   - Outputs: SDB headers
   - Interfaces: HLCL-I/O, HLCL-DBMS, HLCL-Applications

2. Spectral gain correction -

   - Inputs: SDB data
   - Outputs: SDB data plus headers
   - Interfaces: HLCL-I/O, HLCL-Applications

3. Spectral background subtraction -

   - Inputs: SDB data plus headers
   - Outputs: SDB data plus headers
   - Interfaces: HLCL-I/O, HLCL-Applications

4. Spectral summation -

   - Inputs: SDB data plus headers
   - Outputs: SDB data plus headers
   - Interfaces: HLCL-I/O, HLCL-Applications

5. Instrument response determination -

   - Inputs: SDB headers, Instrument response data (FDB)
   - Outputs: SDB headers
   - Interfaces: HLCL-I/O, HLCL-Applications

6. Spectral line and continuum fitting -

   - o Inputs: SDB data plus headers, FDB data
   - o Outputs: SDB data plus headers
   - o Interfaces: HLCL-I/O, HLCL-Applications

7. Spectral deconvolution -

   - o Inputs: SDB data plus headers, Instrument model data, FDB data
   - o Outputs: SDB data plus headers
   - o Interfaces: HLCL-I/O, HLCL-Applications

8. Diffuse source modeling -

   - o Inputs: SDB data plus headers, Instrument model data, FDB data
   - o Outputs: SDB data plus headers
   - o Interfaces: HLCL-I/O, HLCL-Applications

9. Epoch folding -

   - o Inputs: pulsar data headers, pulsar data
   - o Outputs: SDB data plus headers
   - o Interfaces: HLCL-DBMS, HLCL-Applications, Applications-I/O

5.1.3.3 I/O Functions

The HLCL shall maintain an inventory of interactive commands that invoke specific I/O routines via AFEs. The list of specific functions shall be minimally all those provided by the SDB Library and the FDB Library. Document [3] provides a description of each SDB routine; document [4] provides a description of each FDB routine.

In addition, the HLCL shall provide commands for disk and tape I/O with arbitrary files (i.e., other than SDB-type files). These I/O functions shall include opening and closing arbitrary disk and tape files, and reading from and/or writing to opened files. File specifications (format, record length, access mode, etc.) shall be parameters to the HLCL command for opening the file. The variables for reading and writing shall be native HLCL variables.

5.1.3.4 DBMS Access

Three modes of access to the DBMS shall be provided from the HLCL: 1) invoking specific DBMS functions; 2) invoking the native interactive environment of the DBMS; and 3) invoking the DBMS in standalone mode. These are described in sections 4.1.4 and 4.2.3.

5.1.3.5 Batch-Mode Processing

IGORE batch-mode processing shall be implemented by submission of IGORE command files to the IGORE executable image running as a VMS detached processes. IGORE command files shall have the same format as journal files. Command files shall be derivable from journal files or creatable in the editor.

Standard journaling shall not include DBMS operations carried out in the native DBMS environment, thus is shall not be possible to submit in batch mode any journal file that includes invokation of the DBMS enviroment. It shall be possible to submit journal files include access to the DBMS via access mode (1), described in sections 4.1.4 and 4.2.3 (this mode invokes specific DBMS functions from an external program).

It shall be possible in general to distill from the results of a native DBMS environment session the specific DBMS functions that can be alternatively be invoked in access mode (1). When this is possible, journal files that include invokation of the DBMS environment can be edited, and the specific DBMS functions (in the form of HLCL commands) can be inserted. It shall be possible to then submit the edited journal file in batch mode.

5.1.3.6 Environment Initialization

An environment initialization system file shall be executed upon login to the IGORE. Execution shall transfer to an optional user initialization file upon completion of system file.

5.1.3.7 Documentation

Documentation shall consist of an IGORE User's Guide and formal documentation of any software libraries which are part of IGORE. The IGORE's User's Guide shall document all HLCL functions. Documentation of HLCL commands shall also be available as online help. The IGORE User's Guide shall also provide guidelines and instructions for adding user-written FORTRAN programs to IGORE. The IGORE User's Guide shall be under configuration control.

5.2 Input/Output Module

5.2.1 Definitions And General Considerations

The Input/Output (I/O) Module shall consist of all routines which read and write OSSE science data, and AFEs which make appropriate I/O capabilities available in IGORE as interactive commands, procedures, and functions. The I/O Module shall not be an interface to the DBMS or any of the data bases managed by the DBMS.

5.2.2 Data Types Allowed

The I/O Module shall access all OSSE data types as well as auxiliary data.

5.2.3 Input/Output Module Functional Requirements

The requirements given here shall specify the data types that shall be included in the AFEs that interface the I/O routines to the HLCL. The I/O routines specific to OSSE data types are described in documents [3] and [4].

5.2.3.1 SDB I/O

Each SDB routine shall be interfaces to the HLCL via an AFE. All SDB format data and auxiliary data shall be passed across this interface. The representations of these data types in the AFEs are as follows.

1. SDB data:

   - structured headers plus data vectors
   - structure headers only
   - data vectors only

2. Auxiliary data:

   - all VAX FORTRAN data types and structures

5.2.3.2 FDB I/O

Each FDB routine shall be interfaced to the HLCL via an AFE. All FDB format data and auxiliary data shall be passed across this interface. The representations of these data types in the AFEs are as follows.

1. FDB data:

   - structured headers containing arrays of structured variables
   - Auxiliary data:

   - all VAX FORTRAN data types and structures

5.2.3.3 Generic Disk And Tape I/O

The following general disk and tape I/O routines shall be provided:

1. Open. This shall open a disk file. The file specifications (format, record size, access mode, etc.) shall be determined dynamically by passed parameters.

2. Mount. This shall access the VMS mount command. Mount specifications (block size, record length, foreign, etc.) shall be determined dynamically by passed parameters.

3. Read. This shall read from a disk or tape file into a disk file or a suitable HLCL variable. The file to be read and the destination shall be determined dynamically by passed parameters.

4. Write. This shall write to disk or tape file from a disk file or a suitable HLCL variable. The file to be written to and the source shall be determined dynamically by passed parameters.

5. Close. This shall close an opened disk file. The file to be closed shall be determined dynamically by a passed parameter.

6. Dismount. This shall provide access to the VMS dismount command.

5.3 Data Base Management System Module

5.3.1 Definitions And General Considerations

The DBMS Module shall consist of the DBMS "engine" and a set of DBMS applications which interface to the engine. These applications shall have two main functions: 1) to provide the native interactive environment for the DBMS; and 2) to provide an interface through which external programs can call DBMS functions without invoking the native environment of the DBMS. Requirements for the native environment of the DBMS are within the scope of the DBMS specifications. This section describes rudimentary functions of the DBMS that shall be available via its external program interface, and requirements of AFEs that access the DBMS via this interface.

Within IGORE, three modes of access to the DBMS shall be avialable: 1) invoking specific DBMS functions via the DBMS external program interface; 2) invoking the DBMS native interactive environment via the DBMS external program interface; and 3) invoking the DBMS in standalone mode. Mode (3) provides the widest range of DBMS capabilities, but prohibits passing DBMS output directly back to the calling program that invoked the standalone operation. Disk files shall be used to pass output from the DBMS back to IGORE in mode (3). In modes (1) and (2) it shall be possible to pass DBMS output directly back to the calling program across the DBMS external program interface. Output shall always include status and error information; the calling program shall examine this information and, if necessary, signal an IGORE condition (see section 5.5).

Modes (2) and (3) shall require user familiarity with the DBMS; mode (1) shall require user familiarity only with HLCL commands that invoke specific DBMS functions.

5.3.2 Data Types Allowed

The DBMS Module shall access auxiliary data types, and the headers of all OSSE data types.

5.3.3 Data Base Management System Module Functional Requirements

The requirements given here shall specify rudimentary functions of the DBMS and functional requirements of the AFEs that interface the DBMS routines to the HLCL.

5.3.3.1 Rudimentary Functions

The user shall be able invoke the following DBMS operations:

1. Search.
2. Select.
3. Access.
4. Update.
5. Append.
6. Delete.
7. Manipulate.

These operations shall be accessible in any of the three access modes of the DBMS. In modes (2) and (3), these operations shall either be invoked via DBMS query language commands or through DBMS menus or forms.

Output from the DBMS shall be DBMS records or subsets of records. These shall include OSSE data headers, lists of pointers to OSSE data stored in data bases that are not managed by the DBMS (i.e., SDB data, time series data, pulsar data, etc.). Output shall sent directly across the DBMS external program interface in modes (1) and (2), otherwise a disk file shall be necessary if the output is to be accessed by any other module of IGORE. In modes (1) and (2), output shall always include status and error information.

5.3.3.2 AFE Requirments

The following list specifies requirements for AFE access to the DBMS in access modes (1) and (2).

1. Headers shall be represented as FORTRAN structures.

2. Commands in the interactive query language of the DBMS shall be passed to the AFE as character strings. This applies to access mode (1) only.

3. All auxiliary data type shall be represented as VAX FORTRAN data types and structures.

5.4 Applications Module

5.4.1 Definitions And General Considerations

The Applications Module shall be a collection of scientific data analysis programs. It shall include configuration managed code, as well as user-written FORTRAN programs.

5.4.2 Data Types Allowed

The Applications Module shall access auxiliary data types and all OSSE data types.

5.4.3 Applications Module Functional Requirements

Each program in the Applications Module shall interface to the HLCL through an AFE. It shall be possible for users to create an AFE to interface any user-written program to the HLCL.

Minimal functional requirements to be provided by this module shall be those listed in section 2.3.

5.5 IGORE Condition Handler

IGORE shall provide a central error detection and handling facility for all routines in all modules, and an additional separate error detection system for the HLCL/Command Shell. The purpose of the central facility shall be to handle all exceptions encountered in any routine in any of IGORE's modules; it shall be referred to as the IGORE Condition Handler (ICH). The purpose of the HLCL error detection system shall be to handle all errors in usage of the HLCL, including incorrect command syntax and programming structure.

The ICH shall have a core consisting of a condition handler built upon the VAX/VMS Condition Handling Utilities. The condition handler shall be invoked by exceptions detected by the VMS operating system and/or by program-specified conditions which are triggered by software tests explicitly implemented in the offending routine.

The requirements for IGORE's error detection and handling central facility and or HLCL error detection are specified in the following three subsections. The first specifies the functional requirements of the condition handler. The second specifies exceptions that shall require action by the condition handler. The requirements of HLCL

error detection are specified in third subsection.

The term "exception" shall refer to an event that when detected by the VMS operating system causes program execution to be diverted to the VMS exception dispatcher. The term "condition" shall refer to the specific cause of the exception. In the VMS environment, exceptions may be detected by VMS or may be signaled by the program as part of the explicit instructions being executed. The term "condition handler" shall refer to a program which receives control when an exception occurs; it is not part of the normal flow of the routine in which the exception occurs. VMS allows users to supply their own condition handlers. When an exception is detected or signaled, the VMS exception dispatcher invokes one or more condition handlers according to specific rules described in the System Utilities Manual and the Runtime Library Manual.

5.5.1 Functional Requirements Of The Condition Handler

The requirements in this section pertain only to routines which are interfaced to the HLCL via an AFE. These requirements shall apply regardless of the function of the specific routine and the module in which it its contained. The term "application" shall be used to refer to any routine in any module of IGORE, regardless of the routine's function.

5.5.1.1 General Requirements

The general requirements of the IGORE condition handler are as follows:

1. The condition handler shall be built upon utilities provided by the VMS Condition Handling Facility.

2. All IGORE applications shall be interfaced to the HLCL by an AFE.

3. The ICH shall be established in the AFE by placing the address of entry mask of the condition handler in the call frame of the AFE. No other condition handlers shall be established by any subsequently called routines in the course of the application's execution.

4. The ICH shall be de-established immediately prior to return from the AFE to the HLCL, regardless of whether the application executed successfully (i.e., no fatal exceptions) or failed (i.e., fatal exception occured).

5. A configuration managed list of conditions shall be maintained and the condition handler shall be equiped to deal with any and all of them. Conditions on this list shall be referred to as IGORE conditions, and the resulting exceptions shall be referred to as IGORE exceptions. Updating the condition handler to accomodate new conditions shall require only updating the list of configuration managed conditions.

6. IGORE exceptions shall minimally include all exceptions that can be detected by the VMS system without having been signaled. However, it shall not be required that all IGORE conditions result in VMS-detectable exceptions (in the absence of explicit signaling).

7. A disk-based conditions log file shall be maintained to log any exceptions encountered. Entries in the log shall be made by the condition handler according to the specifications below.

5.5.1.2 Exception Detection Modes

Exceptions shall be triggered in one of two modes: 1) VMS system detection, and 2) programmed signaling. The first mode includes all exceptions detected by the VMS system regardless of whether or not the associated condition is tested for in the code. Example: floating point divide by zero.

Programmed signaling shall be the mode for signaling exceptions which are detected by software tests implemented as part of the design of a given routine. Such tests shall therefore be made in the normal course of the program execution. Signaled exceptions shall be the mode for triggering all IGORE exceptions that would not otherwise be detected by VMS. Example: calibration error. This shall also be the mode for user-triggered exceptions.

IGORE shall provide routines for initiating signaled exceptions. These routines shall allow the specification of a message for output and whether the condition is fatal. It shall not be possible to specify as nonfatal any condition that the VMS system treats as fatal.

5.5.1.3 Exception Classes And Associated Condition Handler Response

The IGORE condition handler shall recognize two classes of conditions: continue on warning (COW), and abort on error (AOE). Specific conditions and their classes are listed in the next section.

Continue on warning shall encompass all nonfatal exceptions. The user shall be able to define any condition as nonfatal, except those already classified as fatal by VMS and/or the IGORE system. The course of action taken by the condition handler for a continue on warning shall be as follows:

1. A message shall be output to the terminal and to the conditions log file.

2. Execution of the routine in which the exception was encountered shall continue at the next instruction after the one in which the exception occured.

IGORE shall provide a dynamic switch that turns off COW messages.

Abort on error shall encompass all fatal exceptions. It shall be possible to specify as fatal any exception that VMS and or the IGORE system has classified as nonfatal. The course of action taken by the handler for an abort on error shall be as follows:

1. Further execution of the routine in which the exception occured shall be aborted.

2. If the exception is a fault, it shall be coverted to a trap.

3. A message shall be output to the terminal and to the conditions log file.

4. The VMS system default trace back handler shall be invoked causing traceback information to be output to the screen and to the conditions log file.

5. The call stack shall be unwound back to the driver routine.

6. The IGORE condition handler shall be de-established. (Note that this shall always be done regardless of success or failure of the applictation.)

7. Control shall be transferred back to the HLCL.

Note that when VMS detects an access violation, the exception dispatcher invokes the VMS "catchall handler," which issues message and exits the program. That is, no user-supplied condition handler can recover from an access violation; in such a case, the IGORE image would crash.

5.5.1.4 Message Output

All messages shall be output to the terminal and to the conditions log file. Messages for output shall have four possible sources. The sources and their associated exception detection modes shall be:

1. VMS Message File. This is the default VMS message file invoked by the VMS default condition handlers. The IGORE condition handler shall output an appropriate message from this file for any exception detected by the VMS system.

2. IGORE System File. This shall be the default file of error and warning messages for all IGORE conditions. An appropriate message from this file shall be output whenever such an exception occurs; if the VMS system detected the exception, the appropriate message from the VMS file shall also be output.

3. User File. This shall contain user-defined messages to be output by the condition handler whenever a user-defined program-signaled exception occurs.

4. Immediate Messages. Program-signaled exceptions shall be able to supply messages which are set in the code of the routine that signals the exception. These shall be passed in the exception-signaling utility call either as character strings or character variables.

5.5.2 IGORE Exceptions

IGORE exceptions shall be classified either as continue on warning (COW) or abort on error (AOE) depending on the VMS severity code of the condition. The severity code is determined either by the VMS system component that detected the condition, or in the software-initiated signal. In the first case, IGORE has no control over the severity code. In the second case, the severity code shall be explicitly set when the exception is signaled. All IGORE conditions that must be tested for (i.e., those which VMS will not detect) shall have associated severity codes.

All conditions with severity codes 1, 3, or 0 (success, info, or warning) shall be classified as continue on warning. All conditions with severity codes 2 or 4 (error or severe) shall be classified as abort on error.

Every configuration managed application shall document all conditions which are tested for in its code. All such conditions shall be included in the list of IGORE conditions. All conditions requiring action shall use the utilities provided by IGORE to signal an exception.

5.5.3 HLCL/Command Shell Error Detection

The HLCL shall have its own condition handler. Conditions requiring action by this handler shall be:

1. Syntax errors in HLCL usage.

2. Program structure error.

The HLCL shall include a command that provides the interactive environment or an HLCL program access to the IGORE system condition handler (previous sections). The specific functions of this command shall be:

1. Establish the IGORE condition handler within the interactive environment or within an HLCL program.

2. If established within an HLCL program, specify a branch point in that program if a continue on warning condition occurs.

3. If established within an HLCL program, return to the main-level HLCL and de-establish the IGORE system condition handler if an abort on error condition occurs.

The HLCL shall include a corresponding command to de-establish the IGORE system condition handler as the condition handler of the interactive environment or an HLCL program.

The HLCL shall provide a command to signal an exception, analogous to the utility provided for signaling exceptions within interfaced applications.